GLAVA 1

Neobjektne ekstenzije C++-a

C++ je danas nesporno najmoćniji, ali i najkompleksniji, programski jezik. Zbog kompaktnosti programa, brzine izvršavanja i prenosivosti predstavlja nezamenljiv alat svakog profesionalca. Zbog ovih osobina zauzima dominantno mesto u svetu profesionalnog programiranja.

Jezik C++ se razlikuje od običnog C-a pre svega podrškom objektno orijentisanom programiranju (OOP). Ali, u njemu ima i niz novih mogućnosti, koje nisu objektnog karaktera, zbog kojih je pisanje programa koji nisu objektne prirode udobnije realizovati u C++-u, nego u C-u.

Komentari

Radi pojašnjavanja smisla pojedinih delova programa, može se koristiti komentar. Komentarom se u C-u smatra svaki niz simbola koji počinje parom znakova /* i završava parom znakova */. Na primer,

```
/* U sledecem odeljku upoznacemo se
   sa strukturom C/C++ programa */
```

U C++ postoji još jedan oblik komentara, takozvani jednolinijski komentar. On počinje parom simbola // i proteže se samo do kraja linije.

```
// U sledecem odeljku upoznacemo se sa strukturom C/C++ programa
```

Struktura jednostavnog C/C++ programa

Da biste stekli predstavu o strukturi C/C++ programa analiziraćemo sledeći jednostavan program:

```
#include <stdio.h>
void main()                    // prvi C/C++ program
  {
  int prvi,drugi,rezultat;     // lista celobrojnih promenljivih
  prvi=5;                      // dodela vrednosti promenljivoj prvi
  drugi=3;                  // dodela vrednosti promenljivoj drugi
  rezultat=prvi+drugi;         // izracunavanje zbira
  printf("Zbir %d + %d = %d \n",prvi,drugi,rezultat);    // ispis
  }
```

C/C++ program može se nalaziti u jednom ili više fajlova. Samo jedan od fajlova (glavni programski modul) mora sadržati funkciju **main,** sa kojom započinje izvršavanje programa. Za sada ćemo pisati programe koji se nalaze u jednom fajlu i koji mogu da koriste funkcije iz sistemske biblioteke. U tom slučaju, u programu se mora navesti direktiva pretprocesoru kojoj prethodi znak #. Pretprocesor, kako mu ime kaže, vrši analizu i pripremu programa pre kompilacije (o njemu će biti više reči kasnije).

U navedenom programu direktiva pretprocesoru `#include <stdio.h>` ukazuje da treba uzeti informaciju koja se nalazi u fajlu `stdio.h` (**st**andar**d** **i**nput/**o**utput **h**eader).

Jedina funkcija se morala imenovati sa `main()`. Opis funkcije se sastoji iz ***zaglavlja*** i ***tela funkcije***. Zaglavlje funkcije u ovom najprostijem obliku grade ime funkcije: **main** i zagrade (). Iza zaglavlja navodi se telo funkcije čiji se početak označava znakom: `{`, a kraj znakom: `}`. Unutar ovih zagrada pišu se operatori koji obrazuju telo funkcije. Svaki operator se završava znakom `;`.

Opisni operator

```
int prvi, drugi, rezultat;
```

ukazuje da su promenljive `prvi, drugi, rezultat` celobrojne (`int`).

Operator dodele

```
prvi=5;
```

omogućava dodelu celobrojne vrednosti 5 promenljivoj `prvi`.

Izlazni operator

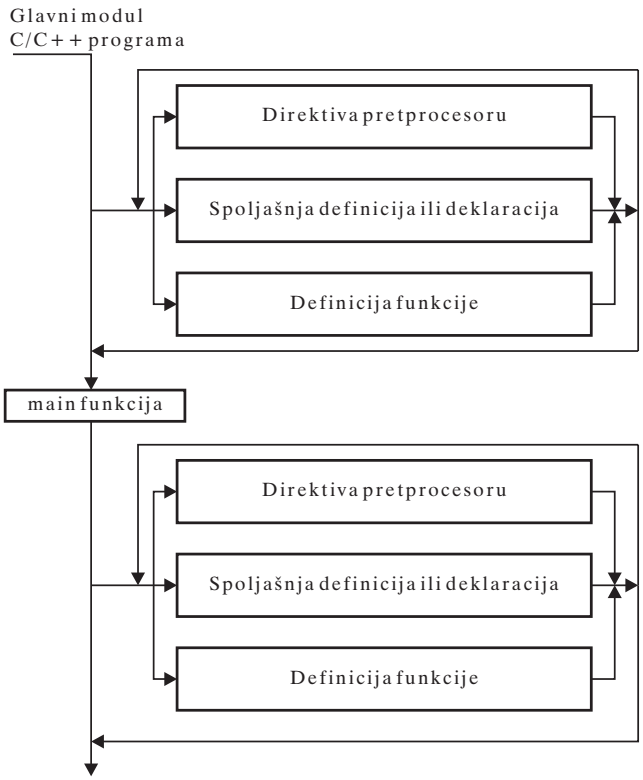
```
printf("Zbir %d + %d = %d \n",prvi,drugi,rezultat);
```

obezbeđuje ispis vrednosti promenljivih `prvi`, `drugi` i njihovog zbira korišćenjem promenljive `rezultat`. Znaci `%d` ukazuju prevodiocu gde i u kom formatu da štampa vrednosti promenljivih `prvi, drugi, rezultat`. Standardnu funkciju `printf()` ćemo razmatrati detaljnije u sledećem poglavlju.

Telo funkcije ima dva osnovna dela: **odeljak sa definicijama i deklaracijama** promenljivih i funkcija, i **odeljak sa operatorima** koji definišu određene akcije. U odeljku za definicije i deklaracije navodi se spisak svih promenljivih koje se koriste i njihov **tip**. **Tipom** se zadaje oblast definisanosti promenljivih, funkcija i izraza koji mu pripadaju, skup operacija koje se mogu izvršiti nad njima i način registrovanja u računaru. U našem prvom programu odeljkom za opis podataka deklarišu se tri promenljive: `prvi, drugi i rezultat`; i precizno je deklarisan tip vrednosti koje im se mogu dodeliti, tj. samo celobrojne (`int`). Izvršavanjem operatora dodele u telu funkcije one dobijaju konkretne celobrojne vrednosti: 5, 3, i 8.

Struktura glavnog modula C/C++ programa može se predstaviti dijagramom na slici 1.

Iz dijagrama se vidi da **main** funkcija ne mora stajati ni na početku ni na kraju programa, kao i da se direktive, spoljašnje definicije i deklaracije mogu nalaziti bilo gde u fajlu.

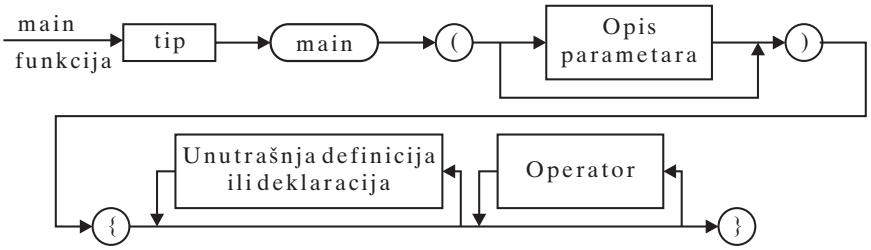


Slika 1.

Glavna funkcija **main** može se predstaviti sintaksnim dijagramom sa slike 2.

C++ dozvoljava deklarisanje promenljivih ne samo na početku već bilo gde u telu funkcije. Najčešće pre prvog pojavljivanja promenljive.

Da bismo videli kako na računaru „radi" napisani program, moramo ga prvo editovati, prevesti, pa tek onda zadati njegovo izvršavanje. Program ćemo testirati korišćenjem konzolne aplikacije Visual C++-a.



Slika 2.

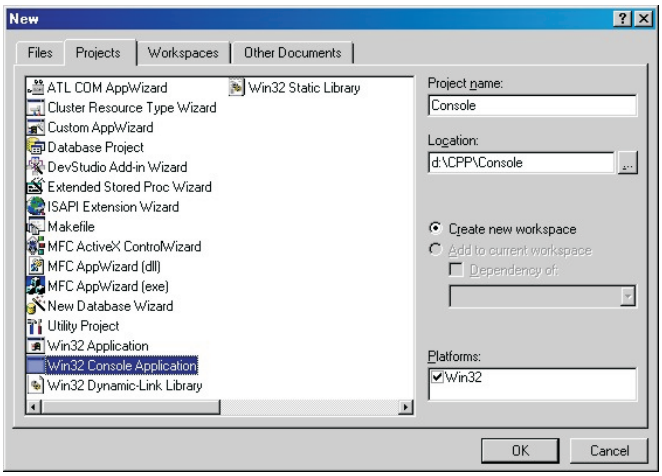
Kreiranje konzolne aplikacije

Konzolne aplikacije i dalje „žive" u Windows programima da bi se programerima koji su i dalje privrženi DOS-u omogućilo da kreiraju i testiraju programe pod kontrolom savremenog operativnog sistema Windows. Drugi razlog za korišćenje konzolne aplikacije je testiranje novih algoritama. Ako ti algoritmi ne zahtevaju prikaz na ekranu grafičke informacije, korišćenje konzolne aplikacije je bolji pristup za proveru njihovog funkcionisanja.

Transformišući program, napisan pod DOS-om, u konzolnu aplikaciju treba imati na umu da je MS-DOS 16-bitni operativni sistem, a Windos 95, Windows 98 i Windows XP 32-bitni operativni sistemi. Zbog toga u **Windows** konzolnim aplikacijama promenljiva tipa **int** zauzima 4 bajta umesto 2 pod DOS-om. To najčešće dovodi do neznatnog rasta memorijskog prostora koji zauzima program.

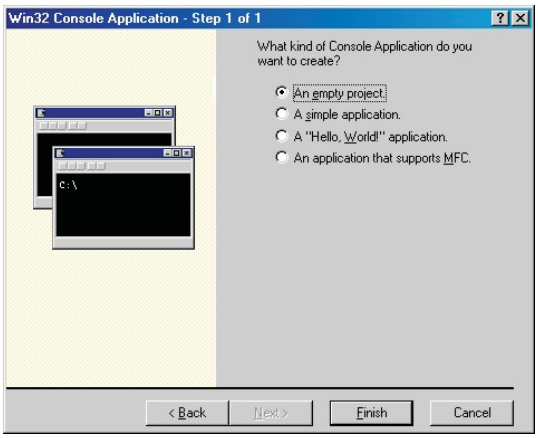
Da biste kreirali konzolnu aplikaciju korišćenjem čarobnjaka, postupite na sledeći način:

1. U Windows-u kreirajte folder, na primer CPP, u kome ćete čuvati C/C++ aplikacije.
2. U Visual C++ (verzija 6 ili 7)-u zatvorite sve otvorene projekte i fajlove, i izaberite komandu **File**, **New**. Pojaviće se dijalog **New** (slika 3) sa otvorenom karticom **Projects** (Projekti).
3. Selektujte opciju **Win32 Console Application** (konzolna aplikacija 32-bitnog Windowsa) i unesite u tekstualno polje **Project name** ime projekta, recimo *Console*. Ovim akcijama dijalog **New** dobija izgled kao na slici 3.



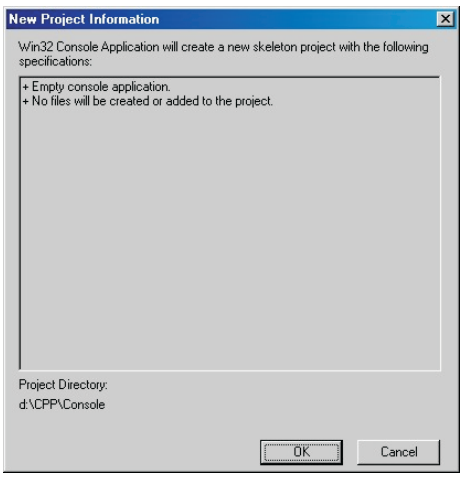
Slika 3.

4. Kliknite na dugme **OK**. Pojaviće se dijalog **Win32 Console Application - Step 1 of 1** kao na slici 4. U radio-grupi **What kind of Console Application do you want to create?** izaberite opciju **An empty project**. Master (ili čarobnjak) će vam kreirati aplikaciju u kojoj nećete imati fajl realizacije, pa je vaš zadatak da ga kreirate.

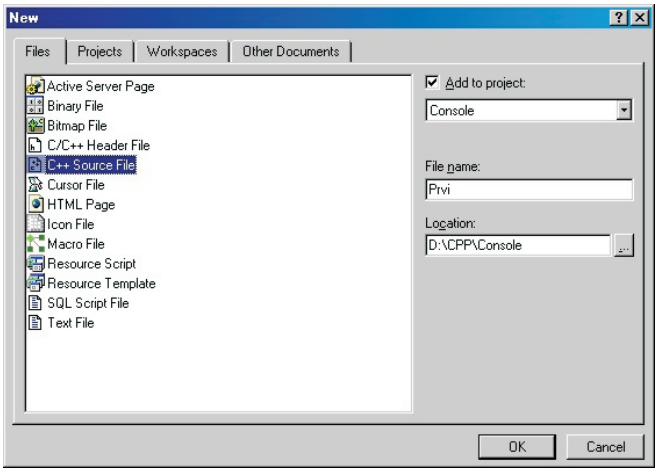


Slika 4.

5. Pošto ste prihvatili ponuđenu opciju, kliknite na dugme **Finish**. Pojaviće se dijalog **New Project Information,** kao na slici 5, u kome ćete dobiti osnovne informacije o svojstvima projekta.

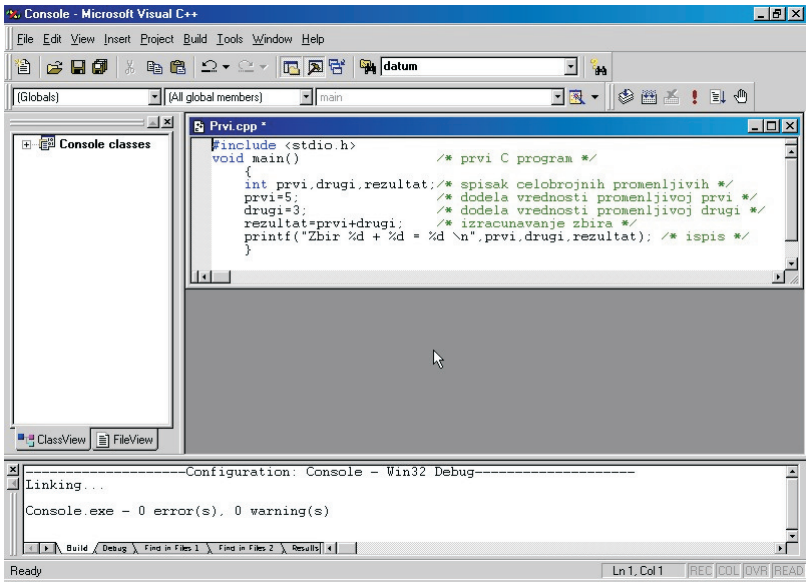


Slika 5.

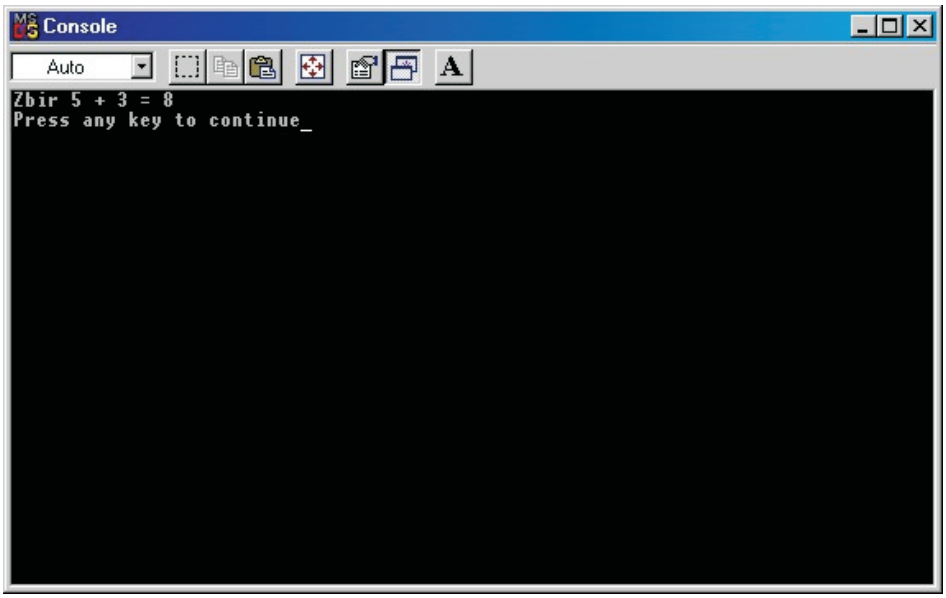


Slika 6.

6. Klikom na OK biće kreiran novi projekat. Da biste projektu dodali fajl realizacije sa C++ programom, izaberite iz glavnog menija komandu **File**, **New**. U dijalogu **New** (slika 6), koji se prikazuje, izaberite *C++ Source File* i u polju **File name** unesite ime programa, na primer: *Prvi*. Klikom na OK zatvara se dijalog i otvara prozor za editovanje fajla.
7. Unesite program koji sabira dva zadata broja (slika 7).



Slika 7.



Slika 8.

Da biste obezbedili izvršavanje unetog programa:

1. Izaberite komandu **Build**, **Execute Console.exe** (Kompilacija, Izvršiti fajl Console.exe) ili pritisnite dugme **Execute Program** na paleti alatki. Pojaviće se dijalog **Microsoft Visual C++,** kojim se saopštava da izvršni fajl date aplikacije ne postoji i traži se da donesete odluku da li da se formira.
2. Klikom na **Yes**, potvrdite da želite kreiranje .exe fajla. Time će doći do linkovanja programa, a nakon njegovog uspešnog završetka započinje izvršavanje. Pojavljuje se MS-DOS program, u kome će se prikazati rezultati izvršavanja programa (slika 8).

String "Press any key to continue", koji se pojavljuje, obaveštava korisnika da je program završio rad i da je prozor ostao na ekranu samo da bi korisnik mogao da pročita rezultat rada programa. Nakon što se uveri da je program korektno izvršen, korisniku je dovoljno da pritisne bilo koju dirku da bi zatvorio MS-DOS prozor i prešao na izvršavanje drugih zadataka.

Da bi se prešlo na izradu nove aplikacije, potrebno je zatvoriti radni prostor komandom **File**, **Close Workspace**. Priprema nove aplikacije odvija se prethodno navedenim postupkom.

Tipovi podataka

Tipovi podataka u C/C++-u mogu biti *osnovni* i *složeni* (ili strukturirani). Osnovni tipovi podataka su: celobrojni, realni ili sa pokretnom tačkom (**floating point**), znakovni, nabrojivi (**enumeration**) i prazan (**void**). Ovi tipovi predstavljaju osnovu za građenje složenih tipova (nizova, struktura, unija itd.) i ne mogu se razlagati na elementarnije tipove. Međutim, ANSI standardom se ne zadaju dijapazoni vrednosti osnovnih tipova, već se samo definiše odnos u bajtima prostora koji zauzimaju, na primer:

```
sizeof(float)<=sizeof(double)<=sizeof(long double)
sizeof(char)<=sizeof(short)<=sizeof(int)<=sizeof(long)
```

Granične vrednosti (INT_MIN, INT_MAX, LONG_MIN, LONG_MAX, ...) za celobrojne tipove zavise od implementacije i sadržane su u fajlu zaglavlja <limits.h> (<climits>). Minimalne i maksimalne vrednosti realnih vrednosti (FLT_MIN, FLT_MAX, ...) su definisane u fajlu <float.h> (<cfloat>).

Logički tip (bool). U C++ za veličine logičkog tipa, mogu se umesto 0 za netačno i ≠0 za tačno, koristiti vrednosti **true** (tačno) i **false** (netačno). Logička konstanta **false** se registruje nulom (0). Svaka druga vrednost se interpretira kao **true**. Kada se **true** konvertuje u celobrojni tip, daje 1. Za opis promenljivih logičkog tipa koristi se rezervisana reč **bool**. Treba da imate na umu da neki stariji C++ kompajleri ne poznaju ovaj tip.

Promenljive

Promenljiva je imenovana oblast memorije u kojoj se čuva podatak određenog tipa. Promenljivu karakterišu ime i vrednost. Ime služi za obraćanje oblasti memorije u kojoj se čuva njena vrednost. U toku izvršavanja programa vrednost promenljive može se menjati .

Opis promenljive, osim tipa i memorijske klase, eksplicitno ili implicitno zadaje i njen *doseg*. On vrlo često ne zavisi od samog opisa, već od mesta u programu gde je naveden.

Doseg identifikatora čine delovi programa u kojima se on može koristiti. U zavisnosti od dosega, promenljiva može biti lokalna ili globalna.

Promenljiva je *lokalna* ako je definisana unutar bloka (blok ograničavaju vitičaste zagrade). Njen doseg je od tačke opisa do kraja bloka uključujući sve umetnute blokove. Promenljiva je *globalna* ako joj se opis nalazi van svih blokova. Doseg takve promenljive je deo fajla u kome je definisana: od tačke opisa do kraja fajla.

Oblast vidljivosti promenljive je deo teksta programa iz koga joj se može pristupiti. Obično se oblast vidljivosti poklapa sa dosegom promenljive. Izuzetak je kada je u umetnutom bloku definisana promenljiva koja od tačke opisa do kraja bloka „zaklanja" istoimenu spoljašnju promenljivu. Tada je spoljašnja promenljiva nedostupna iako joj se doseg proteže na taj deo programa. Globalnoj promenljivoj

se tada može pristupiti korišćenjem operatora za razrešenje oblasti vidljivosti ::(**scope resolution operator –** dvostruka dvotačka). Ovaj operator omogućava korišćenje globalne promenljive i u slučaju kada je „zaklanja" lokalna promenljiva istog imena. Na primer, u segmentu programa:

```
int i=0; // globalna promenljiva
...
int f()
  {
  ...
  int i=0;    // lokalna promenljiva i unutar funkcije
  i++;        // uvecava lokalno i
  ::i++;      // uvecava globalno i
  ...
  }
```

globalna promenljiva **i** postaje vidljiva tek primenom operatora za razrešenje oblasti vidljivosti **::**.

Primer 1. Analizirati vidljivost promenljivih sledećeg programa:

```
#include <iostream.h>
int a=0;              // globalna promenljiva
void main()
  {
  int a=1;            // lokalna promenljiva
    {
    int a=2;          // lokalna promenljiva
    cout << a << endl; // ispisuje vrednost lokalne promenljive a: 2
    cout << ::a << endl; //ispisuje vrednost globalne promenljive a: 0
    }
  cout << a << endl;  // ispisuje vrednost lokalne promenljive a: 1
  cout << ::a << endl; // ispisuje vrednost globalne promenljive a: 0
  }
```

Modifikator const

Pri opisu promenljive može se koristiti modifikator **const,** koji zabranjuje promenu vrednosti promenljive koja joj je inicijalizacijom dodeljena. Takva promenljiva se naziva *imenovana konstanta* ili prosto *konstanta*. Na primer, opisima:

```
int a=1;            // celobrojna promenljiva a
const char c='A';   // znakovna konstanta c
```

definisane su promenljiva **a,** kojoj se vrednost može menjati, i imenovana konstanta **c,** kojoj se vrednost ne može menjati.

Slično statičkim promenljivim, promenljive opisane kao **const** nedostupne su u drugim modulima (fajlovima) projekta. Na primer, ako imamo u projektu sledeća dva modula:

```
// Prvi modul u fajlu mod1.cpp
const float pi=3.14159;

-------------------------------------
```

```
// Drugi modul u fajlu mod2.cpp
#include <stdio.h>
extern float pi;
void main()
  {
  printf("%f", pi);
  }
```

odvojena kompilacija svakog modula biće uspešna, ali linker prilikom povezivanja daje poruku da identifikator **pi** u modulu **mod2.cpp** nije definisan.

C++ kompajler najčešće promenljivu opisanu sa **const**, koja nije lokalna ni u jednom bloku, tretira kao konstantu zadatu direktivom **#define**, tj. prosto vrši zamenu promenljive vrednošću kojom je inicijalizovana. Prednost upotrebe **const** je kontrola slaganja tipova, što može predupredi mnoge greške programa.

Konstante i pokazivači

Značenje opisa **const int x** je jasno: **x** se ne može menjati. Šta onda znači **const int *Ptr**? Moguća su razna tumačenja. To možda znači da **Ptr** pokazuje na fiksnu lokaciju. Ili da se **Ptr** može menjati, a podatak na koji pokazuje ne. Ili znači i jedno i drugo. **const** se može staviti na dva mesta i od toga zavisi značenje. Kada je **const** pre * u deklaraciji tipa, to znači da se podatak na koji **Ptr** pokazuje ne može menjati. Na primer:

```
const int *Ptr;
```

znači da se podatak na koji pokazuje **Ptr** (tj. ***Ptr**) ne može menjati korišćenjem **Ptr**.

Kada se u deklaraciji **const** nalazi iza *, to znači da se pokazivač **Ptr** ne može menjati. Na primer,

```
int *const Ptr=&i;
```

znači da se **Ptr** ne može menjati.

Deklaracija

```
const int *const Ptr=&i;
```

znači da se ***Ptr** i **Ptr** ne mogu menjati.

Spoljašnji opisi i definicije

Svaka funkcija je automatski vidljiva u svim modulima programa. Ako je potrebno da se njena vidljivost ograniči na fajl, u kome je opisana, koristi se modifikator **static**.

Da bi promenljiva ili konstanta bila vidljiva iz više modula, neophodno je:

- da se definiše u samo jednom modulu kao globalna;
- u modulima u kojima treba da bude vidljiva deklariše kao spoljašnja pomoću modifikatora **extern**.

Drugi način je da se promenljiva ili konstanta definišu u fajlu zaglavlja i uključe u module koji će ih koristiti.

Opis promenljive može se dati u formi deklaracije (objave) ili definicije. *Deklaracija* informiše kompajler o tipu promenljive i memorijskoj klasi (**auto**, **extern**, **static**, **register**). Drugim rečima, saopštava kompajleru kako deklarisana promenljiva izgleda i da je ona negde već definisana. *Definicija* predstavlja instrukciju kompajleru da odvoji memorijski prostor u skladu sa tipom promenljive. Promenljiva se deklariše bez definisanja navođenjem ispred opisa rezervisane reči **extern**. Ako opis promenljive sa **extern** uključuje inicijalizaciju, tada se modifikator **extern** ignoriše. U sledećem primeru:

```
int a;          // 1 - globalna promenljiva a
void main()
  {
  int b;        // 2 - lokalna promenljiva b
  extern int x; // 3 - promenljiva x definisana na drugom mestu
  static int c; // 4 - lokalna staticka promenljiva c
  a=1;          // 5 - dodela globalnoj promenljivoj
  int a;        // 6 - lokalna promenljiva a
  a=2;          // 7 - dodela lokalnoj promenljivoj a
  ::a=3;        // 8 - dodela globalnoj promenljivoj a
  }
int x=4;        // 9 - definicija i inicijalizacija x
```

globalna promenljiva **a** je definisana van jedinog bloka *main* finkcije. Memorija joj se dodeljuje u segmentu podataka na početku izvršavanja programa, a doseg joj je ceo program. Međutim, nije vidljiva u celom programu, jer se u liniji 6 definiše istoimena lokalna promenljiva, koja je „zaklanja" do kraja bloka, tj. u oblasti svog delovanja (linije 6-8). Promenljive **b** i **c** su lokalne, oblast vidljivosti im je blok, ali „životni vek" im nije isti: memorijski prostor promenljivoj **b** se dodeljuje u stek memoriji pri ulazu u blok i oslobađa pri izlasku iz bloka, dok se promenljiva **c** registruje u segmentu podataka i živi dok se izvršava program.

U navedenom primeru samo je opis 3 deklaracija, ali ne i definicija promenljive.

Promenljiva se može deklarisati na više mesta u programu, a definisati samo na jednom mestu, pošto deklaracija prosto opisuje svojstva promenljive, a definicija je povezuje sa konkretnom memorijskom oblasti. *Svi opisi iste promenljive moraju biti usaglašeni*.

Sledeći primer ilustruje opis dve globalne promenljive (**a** i **b**) u fajlovima **one.cpp** i **two.cpp** pomoću fajla zaglavlja **my_header.h**.

```
// my_header.h - spoljasnja deklaracija
extern int a;
extern double b;
...
//-----------------------------------------
// one.cpp
#include "my_header.h"
int a;
...
//-----------------------------------------
```